CS 477 Homework #3 Due March 18, midnight

1. The following two exercises use the ITP version (with support for this Java subset) downloadable from the course web site, which has an extension of the list of commands of the ITP specifically designed to support Hoare logic reasoning in our programming language. This version should only be used for verifying Java programs. For all other purposes, for example to verify inductive properties of a functional module, you should still use the version of the ITP that you have used before.

   First unpack the downloaded file. The way this extension works is as follows:

   - start Maude 2.1.1 (the tool needs exactly this version)
   - load into Maude the functional module defining the semantics of our language, `java-es-flat.maude`.
   - define a program (`BlockStatements`) `foo` in a module `foo.maude` importing `java-es-flat.maude` by declaring a constant `foo` and giving an equation defining the constant as the corresponding program text. Also declare a constant `foo-init` which contains all declarations necessary for your program and make sure there are no ("Java") declarations in your program `foo`. The module `foo.maude` should also contain all *auxiliary functions* needed in a proof of correctness of program `foo`. (Note that in the two excercises below this has already been done for you, in the files div2.maude and sumn.maude.)
   - load `foo.maude` (The above mentioned file instead)
   - then load the included version of `itp-tool.maude`
   - then, to prove a Hoare triple

     $$\{P\}\ \texttt{foo}\ \{Q\}$$

     you give a `javax` command, or a `javax-inv` command which also needs an invariant.

   In the homework this command has already been spelled out for you in "sumn.itp" whereas in "div2.itp" the invariant is missing. Before loading "div2.itp" you need to enter your own invariant in the marked place. Then you can interact with the ITP tool to discharge the created first-order goals.

   Note that you can write your solution (i.e. the commands you used to discharge a goal) directly below the javax-inv command and then in one shot load the module which will create the goal and discharge it.

   Here is the sumn.maude module, that you can download es explained above:

```
fmod SUMN-JAVAX is
including JAVAX .
op sum : Int -> Int .
var N : Int .
ceq sum(N) = 0 if N <= 0 .
ceq sum(N) = N + sum(N - 1) if 0 < N .

op sumn : -> BlockStatements .
op sumn-init : -> BlockStatements .
eq sumn =
   'C = #i(1) ; 'X = #i(0) ;
   while ('C <= 'N)
     { 'X = 'X + 'C ;
       'C = 'C + #i(1) ;
   } .
eq sumn-init = (int 'C ; int 'X ; int 'N ;) .
endfm
```

and here is the sumn.itp goal you have to prove (downloadable the same way):

```
select ITP-TOOL .
loop init-itp .

(javax-inv SUMN-JAVAX :
--- specification variables
(N:Int)
--- precondition
(
((int-val(S:WrappedState['N])) = (N:Int)
& (0 <= N:Int) = (true))
 )
--- program
sumn-init
sumn
--- postcondition
(
(int-val(S:WrappedState['X]))
= (sum(N:Int))
)
--- invariant
(
(int-val(S:WrappedState['X]))
= (sum(int-val(S:WrappedState['C]) + -(1)))
&
(1 <= int-val(S:WrappedState['C]))
= (true)
&
((int-val(S:WrappedState['C]) + -(1)) <=
 int-val(S:WrappedState['N]))
= (true)
&
(int-val(S:WrappedState['N]))
= (N:Int))
.)
```

2. Similarly, here is the div2.maude module:

```
fmod DIV2-JAVAX is
including JAVAX .
op div2 : -> BlockStatements .
op div2-init : -> BlockStatements .
eq div2 = 'Y = #i(0) ;
          while (#i(1) < 'X)
            { 'X = 'X - #i(2) ;
              'Y = 'Y + #i(1) ; } .
eq div2-init = (int 'X ; int 'Y ;) .
endfm
```

and here is the goal you should prove, now with the invariant left unspecified; so you have to first come up wiht it, and then prove the goal.

```
select ITP-TOOL .
loop init-itp .
```

```
(javax-inv DIV2-JAVAX :
--- specification constants
(N:Int)
--- precondition
(((S:WrappedState['X]) = (int(N:Int)))
 & ((N:Int >= 0) = (true)))
--- program
div2-init
div2
--- postcondition
((((2 * int-val(S:WrappedState['Y]))
 + int-val(S:WrappedState['X])) = (N:Int))
& ((1 >= int-val(S:WrappedState['X])) = (true))
& ((int-val(S:WrappedState['X]) >= 0) = (true)))
--- invariant
(YOUR INVARIANT HERE)
.)
```